AFRL-IF-RS-TR-2000-107
Final Technical Report
July 2000

A QUALITY OF SERVICE APPROACH TO SURVIVABILITY

BBN Technologies

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. E318/02

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-107 has been reviewed and is approved for publication.

APPROVED:

THOMAS F. LAWRENCE
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

A QUALITY OF SERVICE APPROACH TO SURVIVABILITY

David A. Karr
John A. Zinky
Richard Schantz
Mark Berman
David E. Bakken
William H. Sanders
Michel Cukier

Contractor: BBN Technologies.
Contract Number: F30602-96—C-0315
Effective Date of Contract: 28 August 1996
Contract Expiration Date: 27 August 1999

Short Title of Work: A Quality of Service Approach to Survivability
Period of Work Covered: Aug 96 – Aug 99

Principal Investigator: Richard Schantz
Phone: (617) 873-3550
AFRL Project Engineer: Thomas Lawrence
Phone: (315) 330-2925

Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Thomas Lawrence, AFRL/IFGA, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 2000	Final Aug 96 - Aug 99

4. TITLE AND SUBTITLE

A QUALITY OF SERVICE APPROACH TO SURVIVABILITY

5. FUNDING NUMBERS

C - F30602-96-C-0315
PE - 62301E
PR - D985
TA - 01
WU - 06

6. AUTHOR(S)

David A. Karr, John A. Zinky, Richard Schantz, Mark Berman, David E. Bakken, William H. Sanders, and Michel Cukier

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

BBN Technologies
10 Moulton Street
Cambridge MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFGA
515 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2000-107

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Thomas F. Lawrence/IFGA/(315) 330-2925

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The scope of the project includes creating an interface definition language for Quality of Service (QoS) in support of end-to-end compobility and metrics/models and instrumention for QoS. A design for providing varying degrees of redundancy in an object-oriented distributed system has been completed. Application programmer can specify both a degree of redundancy and the degree of consistency/synchronization among the redundant copies of a resource. The Adaptive Quality of Service Availability (AQuA) project had the objective of controlling the availability of application services in the common Object Request Broker Architecture (CORBA). The project focused primarily on the problem of crash failures, and secondarily on other failures that would cause untimely or incorrect replies to be received by clients. Under these circumstances, AQuA controls the availability of CORBA services by replicating servers, that is, by providing multiple, essentially identical copies of the server operating on different hosts. In this way, the failure of any one copy of the server or of it host tends to be masked by the correct operation of other copies of the server. The AQuA project demonstrated how a distributed system could provide dependability of service. The AQuA release provides dependability of objects (clients as well as servers) by means of replication of the objects. The provision of dependability is highly controllable and adaptable at run time.

14. SUBJECT TERMS

Quality of Service (QoS), distributed systems, dependability, fault tolerance, object-oriented architecture

15. NUMBER OF PAGES

36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

A Quality of Service Approach to Survivability

1 Introduction

Availability is an important property in distributed systems. When a client process within a distributed system issues a service request to a server during normal, correct operation, the client receives a response back from the server after some period of time has elapsed. A service may be said to be ***available*** when the response to a client's request arrives at the client within some nominal waiting-time period after the request was sent. The service should be deemed ***unavailable*** to a client when the client sends a request and, within the longest acceptable waiting-time period, the client receives no reply from the server. While this problem can be caused by delays in the network between client and server or within the server itself, in a well-provisioned system a typical cause for such a problem is that the server process (or the host machine on which it ran) has *crashed*, that is, it has ceased to perform any processing at all.

Recovery from a crash failure typically requires the server (and its host, if that has crashed) to be restarted. If the host has failed due to hardware faults, it may be impossible to restart the server there until the host's hardware has been repaired. In any case, restarting a server is typically a lengthy process, and unless special steps have been taken prior to the crash, important data stored in the crashed server may not be recovered. For example, buffering critical data on a disk drive does not preserve the data during a crash if it is the disk drive itself that crashes.

The Adaptive Quality of Service Availability (AQuA) project [1, 2, 3, 11, 12] began under the umbrella of Quality Objects (QuO) [7, 14] in September 1996, with the objective of controlling the availability of application services in the Common Object Request Broker Architecture (CORBA) [6]. The project focused primarily on the problem of crash failures, and secondarily on other failures that would cause untimely or incorrect replies to be received by clients. Under these circumstances, AQuA controls the availability of CORBA services by *replicating* servers, that is, by providing multiple, essentially identical copies of the server operating on different hosts. In this way, the failure of any one copy of the server or of its host tends to be masked by the correct operation of other copies of the server.

The AQuA team consisted of the following partners:

- The Distributed Systems Department of BBN Corporation [4] a group with depth and breadth of experience in implementing distributed systems, and the prime contractor of the QuO project.

- The Performability Engineering Research Group at the University of Illinois at Urbana-Champaign (UIUC) [10] with expertise in the design and analysis of dependable distributed systems.

- The Ensemble Project at Cornell University [5] a provider of tools for group communication in distributed systems.

Cornell University provided and supported the Ensemble toolkit used in AQuA; additional software to provide the desired CORBA functionality over Ensemble was developed jointly by BBN and the University of Illinois. Furthermore, both BBN and the University of Illinois developed example applications over AQuA.

2 Project History

The AQuA project proceeded in three major phases, each approximately one year in duration.

- During the first year, the design goals of the project were firmed up, a variety of possible implementations were investigated, work was coordinated with the overall QuO project to ensure compatibility between the two projects, and an initial prototype was developed.

- During the second year, the first implementation of the system was completed.
- During the third year, the AQuA system was improved and extended to new capabilities, and the technology was transferred to outside organizations.

The following sections describe this process in more detail.

2.1 The First Year

A series of meetings were held at BBN, Cornell, and UIUC with PIs and developers at BBN and the subcontractors. These meetings identified the components to be used in an initial system implementation, and who would provide them. The initial implementation called for the use of Electra [8], a replication-aware ORB to be supported by Cornell University, with BBN integrating this into the overall QuO architecture.

Additional discussions were held between the AQuA team and potential users of dependable CORBA objects, notably the HiPer-D project at the Naval Surface Warfare Center in Dahlgren, VA.

In addition, UIUC began work to build tools to experimentally validate such systems, and the team at BBN coordinated with the emerging development of the QuO architecture to ensure that the needs of AQuA would be supported in the earliest versions of that architecture.

During the course of the first year, it was found that Electra was unsuitable as a mechanism for AQuA (more detail can be found in the section on Lessons Learned). A prototype was developed that simulated CORBA calls using the Ensemble group communication system from Cornell. This prototype demonstrated the ability to survive and recover from the crash failure of any subset of server replicas, provided that at least one replica of the server survived to preserve its data. Furthermore, using Ensemble, crashed server replicas could be replaced seamlessly by new replicas started on any available hosts in the network, or, when the crashed host became available again, it could be reintegrated. This system was demonstrated in September, 1997.

In addition, a fault injector prototype based on Troll (from the University of Wisconsin, Madison) was developed at the University of Illinois in order to evaluate Ensemble. Crash failures and delays were injected through UDP datagram losses and delays. Faults were injected per host and per link. The preliminary results presented in July 1997 showed that the lack of precision of the injection time led to great imprecision in the observations.

2.2 The Second Year

Following the results of our investigation into Electra, the AQuA team revised its research plan. BBN developed an initial version of a "gateway" over the Ensemble/Maestro system, with support from Cornell. The initial version of this gateway was then transferred to UIUC, and further developed jointly by BBN and UIUC. In addition, UIUC designed and built the first version of Proteus, the AQuA property manager for dependability. (The gateway and Proteus are described in further detail in the section on Implementation Details.)

The AQuA project continued to meet with members of the HiPer-D project during this period. Further, in order to promote the development of commercial ORBs with dependable objects, BBN participated in the development of a Request for Proposals (RFP) for Fault-Tolerant CORBA at the Object Management Group (OMG).

Sample applications were developed at BBN and UIUC to demonstrate various aspects of the AQuA architecture. A slide-show ("Bette") example developed at BBN demonstrated the ability to make traditional RPC calls that passed large data sets (each call retrieved a large image file from the server), and demonstrated the full integration of the dependability property into a QuO contract. A capture-the-flag ("castle") demonstration developed at UIUC showed the dependability management of Proteus, through the crashing and automatic restarting of several replicas of the same replication group and of different replication groups. During the various steps of failure, failure detection, and recovery, the distributed castle application continued to work, showing that Proteus managed dependability in a transparent way. These demonstrations were presented at the Quorum PI meeting in July, 1998.

Following the conclusion about the fault injection experiments performed at the end of year one, and because UIUC could not find a fault injector fulfilling all our needs, UIUC designed a fault injector (called Loki) that would provide more precision in how faults were injected. The design is based on two main ideas. First, fault injection is performed based on the state that different components of the distributed system are in. Second, an analysis is done offline to check if the fault has been properly injected.

2.3 The Third Year

During the third year of AQuA, continuing work on managed bandwidth (previously funded under the Dynamic Integrated Resource Management (DIRM) project) was funded by AQuA, in addition to the ongoing dependability-related design and validation work. Work was also performed on the combination of bandwidth management with dependable (replicated) objects.

A version of AQuA was released in October, 1998 for use with the QuO 1.0 release of September, 1998.

BBN investigated CORBA object references and found issues that complicate the creation of references to objects with QoS attributes, either availability or bandwidth. Steps were taken to overcome some of the problems resulting from the CORBA specification in this area.

BBN integrated a highly-available object (using caching) with QuO instrumentation facilities in order to adapt the use of the server to usage needs at runtime. Specifically, when the server's host was heavily loaded by a higher-priority application, data was taken from the cache even though the server was nominally available. This work was demonstrated at the Quorum/HCC PI meeting in February, 1999.

BBN continued the promotion of commercial development of Fault-Tolerant CORBA, both by speaking on proposals at the OMG and by meeting with proposers at the February PI meeting.

BBN significantly improved the use of object names within AQuA, implementing name-service features on the AQuA gateway to enable replicated objects to find the correct object references at run time rather than requiring static configuration files.

UIUC added a new communication scheme to Proteus able to tolerate crash failures using passive replication. This communication scheme is simpler than the active replication one, since only the leader processes the requests/replies and multicasts its state after each request/reply.

UIUC added an active replication communication scheme and a voter in order to tolerate value faults from the application and/or QuO. The communication scheme is similar to the one developed for crash failures. The main difference is the voter that votes on all messages to check if a majority has been reached. Moreover, the voter stores messages arriving after the majority vote in order to detect potential value faults on all replicas of the group.

UIUC developed a complete interface to the Proteus dependability manager. Using this interface, an application or a QuO programmer can 1) request a particular level of dependability, 2) be notified when that level is no longer met, 3) obtain information concerning hosts managed by Proteus and give advice regarding the hosts on which the manager places replicas, and 4) obtain detailed information regarding decisions that the dependability manager makes, and the faults that it detects.

BBN implemented an RSVP control object over the ACE operating system layer from Washington University in St. Louis, using code developed by our subcontractor at Columbia University.

The AQuA release was fully integrated with the QuO release process and a simultaneous release of both systems occurred in May 1999. Additional AQuA functionality developed through the end of the contract continued to be incorporated via the QuO release process, and will be available with the next QuO release.

UIUC completed an implementation of Loki in August 1999. The current version of Loki integrates the various components of Loki [2] except the parts related to the processing of the measures. A GUI provides the user with an easy way to describe the distributed application and the fault injection campaign and to observe the obtained results.

3 Final Status of the Project

A complete AQuA system has been released and has been transferred to users in the Quorum program. An improved version of this system has been released as part of the overall QuO software release, and again transferred to users. The final version of AQuA developed under this contract will be released as part of the QuO Integration project, and will have the following features, among others:

- Users can replicate either the sender or the recipient of a request (client or server), or both.
- Users can instantiate many distinct replicated objects as part of an application.
- Multiple applications can be configured on the same set of host systems, using the same copy of AQuA.
- A replicated object can send or receive replies from multiple other object instances, including replicated objects.
- Replicated servers can also act as clients.
- Users can create multiple different object instances of the same type, each instance having multiple replicas.
- For each replicated object, a user can specify the type of fault (value and/or crash) to be tolerated by that object, and how many simultaneous faults are to be tolerated.
- Users can make QoS requests to the dependability manager and can receive callbacks regarding the ability of the dependability manager to satisfy the requester's requests.
- Users can "subscribe" to a variety of information used by the dependability manager to make decisions, including information about faults detected and fine-grain information regarding actions taken by the manager.
- Users can receive information regarding the status of hosts that may be used to execute object replicas, and that can be used to make requests regarding which hosts can be used to execute replicas.
- Objects can tolerate crash failure faults or value faults.
- Objects can implement fault tolerance by means of either active or passive replication protocols.

Finally, the operation of the AQuA transport mechanism over wide-area networks by means of reserved-bandwidth channels was investigated, and recommendations have been made for further improvements in this area.

4 Implementation Details

AQuA adds a specialized transport mechanism and property manager to the QuO programming environment.

4.1 The Ensemble Group Communication System

Ensemble is a system for group communication developed at Cornell University. Ensemble provides communication paths with strong semantics over various existing lower-level interprocess communication protocols, such as TCP/IP and ATM. The current implementation of AQuA uses UDP as its low-level protocol.

An Ensemble protocol adds properties to its underlying protocol by adding several "micro-protocol" layers on top of the substrate. These layers optionally can provide some or all of the following properties:

- Group membership, allowing a process to *join* or *leave* a group and to obtain a *group view* with information about the number of members currently in the group.
- Point-to-point messages between two members of a group.
- Multicast messages sent by any one member and received by all other members of the group.
- Sending and receiving arbitrarily large messages at the user level (Ensemble performs the necessary fragmentation and reassembly).
- FIFO ordering of messages, guaranteeing that messages will not be dropped.
- Total ordering of messages, guaranteeing that all group members will see the messages in the same order no matter which members sent the messages.
- *Virtual synchrony*, guaranteeing consistency in message delivery when member processes crash or leave and the group view must be changed.

Furthermore, AQuA was programmed over the Maestro interface to Ensemble, which provides a C++ object interface to support group communication.

4.2 The AQuA Gateway

The AQuA implementation routes CORBA requests through a gateway, which is implemented in C++ over Maestro/Ensemble [13].

Each replica of an object in AQuA is run in its own process over any standard CORBA ORB. In current practice, the ORB used is Visibroker for Java. Along with each object process, a gateway process is run in one-to-one correspondence. The gateway is coded in C and C++ over Maestro/Ensemble. In order to ensure atomicity of failure in case of a host crash, each object replica and its associated gateway process are placed on the same host.

The object processes themselves perform all communication in AQuA via ordinary CORBA Internet Inter-Orb Protocol (IIOP) messages to or from their associated gateway processes. At startup time, the object and its gateway use CORBA to interchange object references: the gateway obtains an object reference to the servant (if any) implemented by the object process, and the object process obtains references to any other objects it might call through AQuA, each such reference routing the request to the gateway.

During subsequent processing, CORBA calls between AQuA objects are initiated by ordinary method calls on CORBA object proxies in client processes, which are transmitted via ordinary CORBA/IIOP protocols to the associated gateway. Within the gateway, an IIOP adapter exchanges appropriate information about the IIOP headers and message bodies with a *dispatcher*, which in turn interacts with *handlers* that implement protocols for communication between replicas of objects and provide fault tolerance. Different handler instances are used for each client-server combination, and different handler types may be used to implement different replication properties, such as active vs. passive replication.

A typical handler implementation is as follows. To implement a replicated object, the gateways associated with the replicas of the object all join a single totally-ordered group (*replication* or *object* group) in Ensemble. There is a separate object group for each object that is replicated in AQuA.

For each possible client-server or peer-to-peer interaction, there is a FIFO-ordered Ensemble group (*connection* group) that includes all replicas of the two objects on either side of the interaction. There is a separate connection group for each possible pair of objects that might interact, and for each direction in which requests might be sent between those two objects.

A request (and the corresponding reply, if any) therefore passes through at least two gateways, one associated with the client and one associated with the server. Several handlers have been developed. Each handler is specific to a communication scheme (cf., Section 5.5). In the case of active replication tolerating crash failures, the client-side gateways collate copies of the request (if the client is replicated) using the client's object group, and forward the request via the appropriate connection group so that it is received by the servers. A final retransmission of each request over the server's object group ensures that each server replica receives requests in identical order even if they were sent by completely different clients.

Failure of one copy of an object will cause the gateways of other objects to resend buffered messages as needed to prevent failure of any requests in progress. As requests complete, the buffers used for their retransmission can be deleted.

4.3 The Proteus Dependability Manager

The property manager in AQuA is Proteus [12], a system that manages dependability of objects by means of replication. A QuO contract associated with an AQuA process or an application process itself can issue requests to Proteus requesting that a named service be provided with the ability to tolerate a specified number of faults of a specified type. The fault types that may currently be tolerated are faults due to crash failure of a process and faults due to incorrect values being returned by a single process in response to a service call. Proteus informs the QuO contract or application that made the dependability request when the requested property is satisfied, and when it is not satisfied, so that the contract may make appropriate transitions between operating regions.

Proteus satisfies property requests by creating a sufficient number of replicas of the requested object, each replica on a different host. The list of permissible hosts to use is read by Proteus at startup time and can be temporarily restricted at any time during Proteus's execution. The Proteus dependability manager and factories use their own AQuA gateways to communicate; specialized handlers in these gateways provide highly scalable communication between these components and other AQuA applications, and gather data on AQuA applications as required by Proteus.

In order to run Proteus, a dependability manager process must be started, and on each host where replicas of user objects might run, a Proteus factory process must be started. In the current implementation, the dependability manager is not replicated and is not itself dependable, although user applications will continue to function after failure of the dependability manager as long as replicas of those objects exist. Since the Proteus processes themselves all use AQuA gateways for communication, however, several options exist for future work to make Proteus itself dependable.

As part of the AQuA system, Proteus implements extremely flexible runtime control over the dependable operation of a replicated object. The desired level of fault tolerance can be adjusted upward or downward after the object has been started; as this is done, replicas of the object will be created or destroyed seamlessly (without interfering with the stored data or continued functioning of the object) as required to maintain the requested level of fault tolerance. However, there may not be enough hosts available to run the replicas required to support the requested level of fault tolerance. When the request cannot be fulfilled (either when requested, or during a later period of time), Proteus alerts the QuO contract so that other corrective action, if any, may be taken.

The selection of permissible hosts where replicas may be run is also variable at run time. Users may dynamically restrict processing to prevent it from occurring on certain suspect hosts, or may lift such restrictions, during the object's processing. Restriction of a host on which an object replica resides will cause that replica to be eliminated from its replication group, and replaced by a new replica on a permitted host, if any is available.

4.4 The Loki Fault Injector

UIUC developed a validation approach for the AQuA architecture using fault injection. However, UIUC could not find an existing fault injector adequate for the architecture. Existing fault injectors do not allow an accurate fault injection based on the state of specific parts of the system. Moreover, no existing fault injector checked afterwards to see if the injection was properly done. These two deficiencies led to the decision to build a new fault injector, which we call Loki.

Loki injects faults based on the partial view of the global state of the distributed application. The idea is that a fault injector for distributed systems must be aware of the states of different parts of the system (not just the state of the local node), without knowing the state of all the parts of the system. The local state on a node is described using a state machine. The various nodes are connected through a network in order to communicate the local state to the nodes that need it to build their partial view of the global state. The fault injection depends on that partial view of the global state. However, because of the delay needed to exchange a message, the fault injection might occur when at least one of the nodes in the partial view is already in a new state. Each injection must thus be checked. This is done off-line, where the events from each node are projected on a single timeline. A software clock synchronization approach has been used to order events from different nodes on one timeline. Moreover, physical bounds are calculated for each event, ensuring that the event really occurred sometime in that interval. After being projected on the single timeline, the ordering and overlapping of these event intervals are analyzed to verify that the fault was properly injected. Finally, statistical features are used to calculate various measures resulting from the fault injection campaign. For more details, see [2].

5 Research Results

5.1 CORBA Standards for Names and References

As a general rule, the common denominator of all CORBA reference schemes is the Interoperable Object Reference (IOR). An IOR uniquely identifies a servant that provides an active (currently running) implementation of some object. This identification includes three essential pieces of data:

- The IP address of the host on which the servant is running.
- The port number of a port on the servant's host to which requests to the servant should be sent.
- An *object key*. The object key is used by the process running the servant to identify that servant uniquely among all possible servants that the process might run.

There are facilities in CORBA for a client object to gain access to a servant even when the client does not yet hold a correct IOR. For example, a process receiving a request on a service it no longer implements can return the client a forwarding address for that service (in essence, the IOR of a new servant implementing the service elsewhere), although this scheme does not work if the old servant's process has simply crashed. Alternatively, a client can refer to a service by name, and access the named service with the help of a CORBA name server. This scheme requires the servant's process to register the correct IOR for the servant under the correct name.

There is no special support in CORBA to ensure that the servant returned by the name server at any given time is an accurate representative of the servant returned to a previous request. That is, suppose the old servant has crashed (or has shut down silently) and a new servant has been created and registered with the name server. The name server cannot tell a client whether the new servant is a proper continuation of the old servant (that is, it has recovered the essential state information from the old servant) or whether it is essentially a completely new instance of the service (having lost all information that was recorded by the old service).

There is not yet support in CORBA for various standard techniques for using process replication to make services fault-tolerant. A standards process aimed at this goal is under way at the OMG as of August, 1999. We expect the outcome of this process to be a set of commercial toolkits that allow CORBA calls to be made to or from replicated objects, most likely with the proviso that both client and server must be coded to the toolkit. The toolkits for fault-tolerant operation are very likely to be more or less incompatible at first with other added-value features that may be desired by CORBA programmers.

Some of the difficulty of fault-tolerant CORBA processing lies in the very nature of the IOR. Many forms of replication require that the client's request be delivered to all replicas of the server, not just to one address. When one replica of the server fails due to a host crash (a typical scenario to be handled by fault-tolerant CORBA), there cannot be a forwarding agent listening to the port specified in the IOR of the failed replica. What a client of a replicated service needs in such cases is not the address merely of a single replica, but the address of a *group*, an identifier that is mapped (via the appropriate services; in AQuA the Ensemble "gossip" name server is used) to the set of all replicas currently in existence, a mapping that is changed from time to time as replicas disappear or are added to the group.

The approach to this problem in AQuA was to interpose gateways between clients and servers. In this scheme, a (possibly replicated) CORBA object accesses all other (possibly replicated) CORBA objects through a gateway process located on the client object's host. The client object holds an IOR for each service in this scheme, but all such IORs point to the same IP address and port number, where the gateway is listening. The object keys in these IORs are plain text strings to which the gateway attaches prefixes and suffixes according to fixed rules to obtain a group name in the Ensemble group communication system. Ensemble is then used to distribute copies of requests to all group members (gateways associated with each replica of the server), which forward the requests to the actual IORs used by the actual servant replicas. Thus, although CORBA requests use some GIOP features (for example, CDR formatting of the request body) throughout a remote method call, the actual IIOP protocol is used only locally (between CORBA object replicas and their own associated gateways), while Ensemble is used for all distributed communication between hosts. Of course, when it is known that neither client nor server is replicated for fault tolerance, the gateway need not be used, and requests may be sent using ordinary CORBA techniques, even from a client that at other times makes requests to replicated servers.

The use of gateways eliminates many of the failures that would otherwise be associated with IORs held by CORBA clients, but introduces an important problem of its own, namely that an object reference held by one object (that is, an IOR pointing to its own gateway) cannot be used by any other object. In order to access other objects through the gateway, a client must be aware that either itself or the desired server is replicated for fault tolerance, and must send a request to its own gateway for the server named by a particular plain text string. The gateway then acts as a private naming server for this client, returning an IOR (pointing to the gateway itself) that should be used to send requests to be forwarded to the server. CORBA provides no support for passing such references correctly to other processes, though of course application-specific code may work around this problem by passing the plain text strings that other objects should use in requests on their own gateways in the current version of AQuA.

5.2 Replication of Objects over a Wide-Area Network

Using group communication or other mechanisms to support active replication is typically not considered practical over a wide-area network (WAN), because frequently the bandwidth between group members is low and the variance in message latency is high. Both of these phenomena make it difficult to maintain a group with a consistent membership list and consistent application state.

BBN Technologies investigated the feasibility of placing the replicas of a fault-tolerant object on hosts distributed over a WAN. The test configuration was to send Ensemble traffic using UDP as the transport between two hosts on different LANs connected by a WAN. The WAN consisted of two Cisco routers connected by a 1.3Mbps serial link. In addition to the Ensemble traffic, a competing test program generated cross traffic in two separate flows in sufficient volume to completely consume all the available link bandwidth. The routers had a buffer size of 64 packets. Tests were performed sending various size Ensemble messages under two sets of conditions: in the "RSVP" case, the routers would reserve 1.0Mbps bandwidth for the Ensemble flow using RSVP; and in the "no-RSVP" case, no reservation was made for Ensemble traffic. The routers used a fair-queuing algorithm, so in either case the Ensemble traffic was guaranteed to get some of the available bandwidth.

Our tests uncovered two important properties of Ensemble over RSVP:

1. Ensemble is very sensitive to the amount of buffering in the routers. When the Ensemble message size got over a certain size, a "congestion collapse" occurred and the effective throughput of Ensemble dropped quickly.

2. RSVP bandwidth reservation improved Ensemble throughput, i.e., isolated the Ensemble traffic from the cross traffic. But RSVP did not improve throughput for message sizes beyond the congestion collapse.

Figure 1 shows the results of two network configurations with and without RSVP reservations. For both the RSVP and no-RSVP cases, the throughput increased as the message size increased up to the point where the router buffers

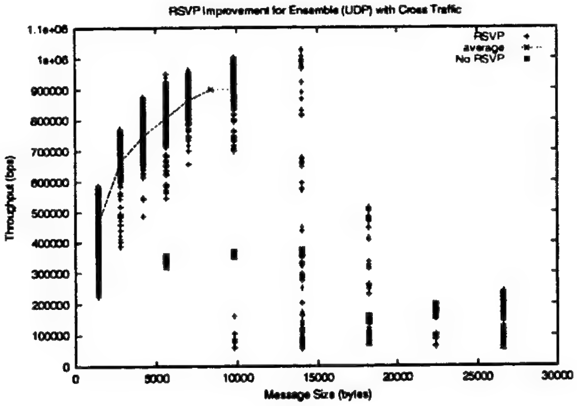

Figure 1: Performance of Ensemble over RSVP

were full. At this point the router would start dropping packets. When a single message (approximately 1400 bytes in these configurations) was dropped, Ensemble would interpret this as a transmission failure of the entire Ensemble message (hundreds of kilobytes in some cases), and would retransmit the entire message, including packets previously transmitted successfully. Thus, the throughput dropped drastically.

The RSVP flow did get more throughput below the congestion collapse point because the RSVP reservation allowed more Ensemble messages through than cross traffic. The congestion collapse point for RSVP and no-RSVP is about the same because the routers' buffers are evenly divided between flows, regardless of the RSVP reservation (ignoring the RSVP burst size).

These results have several implications for future work with AQuA over a WAN. The first is that RSVP was successful in maintaining Ensemble group communication at a fairly high level of throughput in the face of heavy competing traffic flows. The second is that additional work will be required to support AQuA applications that have very large requests or replies, such as when a large object is passed by value. Provision must then be made either in the Ensemble stack or in the AQuA gateway's handlers to control data flow so as to avoid the congestion collapse point, if necessary by breaking up large requests or replies into segments. In addition, a small re-implementation within the Ensemble stack should be able to eliminate or reduce the need for re-transmission of packets that were not dropped.

5.3 Applications to Intrusion Detection

As the result of development efforts at UIUC and BBN, it proved possible to use AQuA to monitor the possible effects of a hostile intrusion into the system under its control as well as to maintain dependable services. Specifically, Proteus detects and provides an interface for other objects to monitor numerous process events that are of interest to intrusion detection systems. BBN was able to exploit this synergy to integrate AQuA into a demonstration intrusion detection system under the QuO effort, and to incorporate this demonstration into the QuO release under the name AQuA IDS (aquaids).

5.4 Design of Multiple Groups

In the AQuA architecture, the basic unit of replication is a three-process pair, consisting of an application, gateway and QuO runtime. A basic replication unit thus contains several distributed objects, but we refer to it as an AQuA object. Furthermore, when we say that an "object joins a group" we mean that the gateway process of the object joins the group. Mechanisms are provided to ensure that if one of the processes in the object crashes, the others are killed, thus allowing us to consider the object as a single entity that we want to make dependable. Four group types are used in the AQuA architecture: replication groups, connection groups, PCS (Proteus Communication Service) group, and point-to-point groups.

A replication group is composed of one or more replicas of an AQuA object. A replication group has one object that is designated as its leader and may perform special functions. Each object in the group has the capacity to become the object group leader, and a protocol is provided to make sure that a new leader is elected when the current leader fails. To maintain a group, Ensemble uses protocols that use group leaders. For implementation simplicity, the object whose gateway process is the Ensemble group leader is designated the leader of the replication group. This allows Proteus to use the Ensemble leader election service to elect a new leader if the object leader fails. A connection group is a group consisting of the members of two replication groups that wish to communicate. A message is multicast within a connection group in order to send a message from one replication group to another replication group. Reliable multicast to the dependability manager is achieved using the Proteus Communication Service (PCS) group. The PCS group consists of all the dependability manager replicas in the system. The PCS group also has transient members. These transient members are object factories, AQuA applications, or QuO objects that want to multicast messages to the dependability manager replicas. Through the PCS group, AQuA applications provide notification of view changes, QuO makes requests for QoS, and object factories respond to start and kill commands and provide host load updates. A point-to-point group is used to send messages from a dependability manager to an object factory. Each object factory is in its own point-to-point group. When the dependability manager wishes to send a message to the receiving object factory, the dependability manager joins the group of that object factory.

The choice of having multiple groups is a design choice made to avoid having to manage one large group. A view change concerning any member of a large group would request to freeze all other members of the group. Moreover, if their tasks are very different, some members that are frozen might be only indirectly concerned by the view change. We chose to define a collection of group types, each one having a different role. Therefore, a view change will freeze a smaller number of objects. Moreover, the objects in each group type have a similar task. Finally, this group structure allows any two objects to communicate using the advantages of group communication systems without having one large group.

5.5 Tolerating Crash and Value Faults using Active and Passive Replication

The AQuA architecture has been designed to tolerate different fault types (i.e., crash failures, value and time faults) using various replication schemes (i.e., active and passive). UIUC has implemented three different schemes in Proteus:

- Toleration of crash failures using active replication and a voter with a pass first policy.
- Toleration of crash failures using passive replication.
- Toleration of value faults and crash failures using active replication and a voter with a majority policy.

For each case, a specific communication scheme has been designed and implemented. In addition, in some cases, a specific voter has been developed. A simple voter with a pass first policy is used for tolerating crash failures using active replication. Since only the leader processes the requests/replies, there is no voter when using passive replication. Finally, for tolerating value faults and crash failures, a voter has been implemented on each replica to vote on the received messages in order to reach a majority value.

Let us now detail the three communication schemes and the voter associated with the scheme. Let $O_{i,k}$ be replica k of replication group i, and let object $O_{i,0}$ be the leader of the group. Suppose that replication group i is the sender group and group j the receiver group. To send a request to the object replicas $O_{j,k}$, the communication steps vary depending on whether only crash failures have to be tolerated, or if value faults (and crash failures) have to be tolerated.

5.5.1 Active Replication for Tolerating Crash Failures

The first communication step consists of all objects $O_{i,k}$ using reliable point-to-point communication to send the request to $O_{i,0}$ (step 1). The leader then multicasts the request in the connection group (step 2). Since there can be multiple replication groups, in order to maintain total ordering of all messages within the replication group, $O_{j,0}$ multicasts the message again in replication group j (step 3). After processing the request, all objects $O_{j,k}$ send the result through a point-to-point communication to $O_{j,0}$ (step 4). The same set of steps used to transmit the request is then used to communicate the reply from replication group j to group i. Steps (5) and (6), which are responsible for transmitting the reply, are similar to steps (2) and (3) respectively.

Each replica has a voter, but only the voters of the replication group leaders are active. All messages received by the leader go through the voter. The task of the voter is to multicast the first message it receives.

5.5.2 Passive Replication for Tolerating Crash Failures

The first communication step prescribes that the leader $O_{i,0}$ multicast the request and the state in replication group i (step 1). The leader then multicasts the request in the connection group (step 2). After processing the request, the leader $O_{j,0}$ multicasts the result and the state in replication group j (step 3). Finally, the leader multicasts the reply in the connection group (step 4).

No voter is used in passive replication since only the leader processes the requests/replies.

5.5.3 Active Replication for Tolerating Crash and Value Faults

The first step consists of all objects $O_{i,k}$ multicasting the request in replication group i (step 1). The leader then multicasts the request in the connection group (step 2). Since there can be multiple replication groups, in order to maintain total ordering of all messages within the replication group, $O_{j,0}$ multicasts the message again in replication group j (step 3). After processing the request, all objects $O_{j,k}$ multicast the result in replication group j (step 4). The same set of steps used to transmit the request is then used to communicate the reply from replication group j to group i. Steps (5) and (6), which are responsible for transmitting the reply, are similar to steps (2) and (3) respectively.

Each replica has a voter. Depending on the number of faults to tolerate, the corresponding required majority is calculated. Each voter checks each received message to see if the number of messages agreeing on the content have reached the majority. If the majority is reached, the replica multicasts the majority value. However, the voter continues to store the messages corresponding to the sequence number of the majority value. This is done in order to detect all value faults. When the replica receives a message whose sequence number is such that no other message with the sequence number of the majority value can be received, the voter checks the content of the former received messages and communicates to the dependability manager the names of the replicas that have sent messages whose values are different from the majority value.

5.6 Use of Partial View of the Global State in Fault Injection

The concept of partial view of a global state has been introduced during the design of Loki. This original concept is fundamental for accurate fault injections in a distributed application. In most distributed systems, each component only knows the state of the other components in the system at the time of synchronized events. At other times, knowledge of the state of any given component by other components is not guaranteed. In order to evaluate and remove faults in a fault-tolerant distributed system, it may be necessary to inject faults in a component based on the state of one or more other components, even at a time when the complete state of the system is unknown. To inject such faults, it is not necessary to know the global state of the system at all times; instead, a "partial view" of the global state of the system is sufficient. Informally, we mean the state of "interesting" components and knowledge of "interesting" state changes. The portion of state that is interesting depends on the particular system being studied and the faults one desires to inject.

With this in mind, a distributed system can be subdivided into the pieces that are necessary for a given fault injection campaign. These pieces, which include both the portions of the distributed system into which faults are to be injected and the parts from which state information must be obtained, are called *nodes.* The Loki run-time architecture provides a framework for maintaining a partial view of the global state using a set of nodes, and for conducting fault injections using that partial view of the global state. More specifically, Loki maintains a partial view of the global state by using state machines associated with nodes that communicate with one another. Each node in the system is coupled with a state machine that tracks the state of that node relevant to the fault injection campaign. These state machines send state change notifications (messages containing information about the global state of the system) to subsets of all of the state machines in the system as necessary to maintain the partial view of the global state needed for fault injection.

Because Loki is designed to be as non-intrusive to the application as possible, it does not block the system under study while waiting for a notification to arrive at another state machine. Accordingly, a system may actually change state again by the time a state change notification reaches its target state machine(s). This implies that the partial view of the global state seen by Loki is not always correct. The correctness of a particular state is determined by the holding time of the global state and the time needed to transmit a notification after the state is reached. Any fault injection based on a global state of the system must thus be checked after the experiment is completed, to see whether the fault was injected as intended.

In summary, the two central innovations of Loki are the provision of a mechanism for recording and sharing a partial view of the global state between nodes in a distributed system, and the development and application of a theory to determine whether a fault injection that depended on the partial view of the system global state was done correctly. Together these give us the ability to inject correlated faults into a distributed system efficiently and predictably. To the best of our knowledge, it is the first successful attempt to provide this functionality. Furthermore, as shown by the experimental results that we have obtained [2], we can successfully inject an increasing percentage of intended faults as the time spent in a chosen state increases. More importantly, we can identify the experiments that led to successful injections, and use these experiments in fault removal and/or dependability assessment activities.

6 Lessons Learned

The AQuA team learned a number of lessons during the course of the AQuA project. In this section, we discuss the results obtained, arranged by topic.

6.1 Replicated CORBA Objects: a Nascent Technology

We learned that adequate technology to support replicated objects under CORBA is not yet generally available. This conclusion is more pessimistic than the expectations at the outset of the AQuA project, when it was expected that such technology would prove to be available either at the beginning of the project or during the first year.

6.1.1 The Demise of Electra

At the outset of the AQuA project, a system known as Electra had already been developed at Zurich University in Switzerland (and further refined at Cornell University) to support CORBA requests on replicated objects. Electra, however, requires an underlying mechanism to implement group membership and multicasting of messages, and at that time the only such mechanism in use was Isis, an older group communication system for which support was being withdrawn. In theory, Electra was supposed to be portable over other group communication mechanisms such as Ensemble; in practice, however, Electra proved to make hidden assumptions about group membership properties that were not supported by Ensemble. Worse still, while Electra provided support for requests made to a replicated object, it provided no apparent support for requests made *from* a replicated object, as might occur in a distributed system having more than two layers in the call tree.

As a result, after an extensive investigation, we decided to abandon plans to use Electra in the AQuA project. A similar decision was made by the Nile project at the University of Texas, whose researchers rejected plans to port their software from Isis to Electra over Ensemble.

6.1.2 Slow but Steady Progress at the OMG

We expected (and continue to expect) that commercial vendors within the OMG will support replication of CORBA objects. ORBs supporting replication have been implemented, most notably Eternal [9]. BBN was directly involved in the call for proposals and later stages in the development of a CORBA standard for fault tolerance (availability), beginning in December 1997. As of August 1999, a standard is under development, but at the request of the vendors involved, the vote on the standard has been postponed multiple times. As a result, no CORBA standard product is available at this time, although preliminary implementations (which may or may not meet the eventual standard) exist.

6.2 The Usefulness of Gateways

The gateway concept, as recognized in CORBA standards, proved to be a very useful research and development tool. This technology enabled us to implement both AQuA replicated objects and (in separately-funded work) CORBA objects over links whose bandwidth was managed by RSVP, using a common core of technology to interface to and adapt CORBA protocols in both projects. Further, this approach promoted greater use of commercial off-the-shelf (COTS) software, because it allowed clients and servers to run over standard ORBs such as Visibroker (with QuO layers added as necessary) rather than requiring a completely new specialized ORB (which would almost certainly be relatively inflexible) to provide properties such as dependability.

6.3 More Work Needed on Wide Area Networks

While our investigations into wide-area transport showed promise, they also showed that more work is required in this area. Group communication over the wide area has long been disparaged as impractical, and so development in that area appears to have lagged. Reservations via RSVP appear to be capable of improving the wide-area performance of group protocols substantially, but some additional development of the protocols (e.g., the Ensemble stack) appears to be called for if further progress is to be made in this area.

6.4 Lack of Fault Tolerance in Group Communication Systems

Most group communication systems, including Ensemble, are based on the assumption that processes fail by crashing, but no mechanism is implemented to ensure that processes fail only by crashing. Furthermore, recovery by automatically starting new processes on the same or different hosts is not implemented in the protocol stack. Instead, it is left to the application. A fault tolerance framework is thus necessary to tolerate other fault types and provide more sophisticated recovery mechanisms than process exclusion. The framework could be implemented at the process level by implementing further fault tolerance in Ensemble. However, in order to be independent of any particular group communication system and to fully use the features offered by CORBA applications, we have provided additional fault tolerance above the group communication infrastructure. The framework we have developed is able to tolerate crash failures of processes, as well as value and time faults of CORBA objects.

7 Future Work

A number of research directions that were begun under the AQuA project are being pursued, or have great potential to be pursued fruitfully, under future funding efforts, including the Quorum Distributed Object Integration project.

7.1 Integration of an ORB in the Gateway

BBN has made investigation into replacing the internally-developed CORBA module of the various QuO gateways (including AQuA's) with portions of the TAO ORB from Washington University at St. Louis. This development has only recently become possible because of more open interfaces in CORBA, by which the gateway might use the ORB in its own process space to obtain a copy of a CORBA request (from a client's ORB) for forwarding, rather than (inappropriately) unmarshalling the request in the gateway. This represents a significant improvement over the previous "interceptor" standard, which was inadequate for the needs of AQuA as well as other QuO applications.

The use of TAO also promises to enable a standards-based integration of the AQuA gateway handlers into the object processes themselves, that is, to replace the current CORBA calls to gateways with in-process method calls. A non-TAO prototype of such a system was constructed at UIUC, and the technology is expected to reduce the overhead of AQuA remote method calls.

7.2 Replication over Wide-Area Networks

As described in the previous section on Lessons Learned, preliminary investigations into replication over a wide-area network indicate that QuO has the potential to improve performance in this area. Due to the underdeveloped nature of previously existing software in this area, however, additional work will be required if this potential is to be fully developed.

7.3 CORBA Object References

The section on Lessons Learned described shortcomings of basic CORBA object references. Work is in progress within the ongoing Quorum Distributed Object integration project to use the flexibility of the QuO architecture to overcome these shortcomings.

7.4 Port AQuA Architecture to NT

This work would result in a release of the AQuA components, including the Proteus dependability manager, the factories, and the gateways that run on NT. The current release is for Linux.

7.5 Evaluation and Enhancement of Current Architecture Performance

This work would provide testing and analysis to determine precisely the current performance of different parts of the AQuA components, to determine AQuA's current performance bottlenecks, and to investigate ways to improve its performance.

7.6 Make the Proteus Dependability Middleware More Dependable

In the current implementation, an application will continue to run dependably if the dependability manager or a factory fails, but the dependability manager or factory will not restart. This work would make the dependability manager and factories themselves more dependable in servicing applications using AQuA.

7.7 Provide Additional Replication Schemes Beyond Active and Passive Replication

The AQuA architecture currently supports passive and active replication. In addition to these basic strategies, there are other strategies that have particular performance and dependability properties that are useful in certain circumstances. This work would add more replication protocols, giving us the ability to support a wider range of QoS requests.

7.8 Runtime Switching between Replication Schemes

This work would develop mechanisms that support switching between certain replication schemes while an application is running. This capability is useful for applications whose dependability and resource utilization profile requirements change or need to adapt to changing operating conditions at runtime. The work involves finding "changepoints," where one can safely change from one scheme to another, and coordinating stopping the old mechanisms with starting the new ones.

7.9 Mechanisms to Switch between Dependability Strategies at QuO Level

This work, which complements the previous item, would investigate adaptation strategies that can make use of the capabilities developed in the previous two items.

7.10 Sophisticated Interface to the Dependability Manager

This work would continue to enhance dependability management interfaces in order to allow other resource managers and resource status information services to provide information to Proteus useful for configuring for dependability, and in providing dependability information to other services. It would enhance the QoS request interface to allow more precise specification of dependability requirements in order to select among the additional replication strategies that will be developed.

7.11 Toleration of Time Faults

This work would investigate the tolerance of timing faults within the AQuA architecture. In particular, it would provide mechanisms to detect when a particular replica in a replication group is responding too slowly, and develop dependability manager policies to diagnose and recover from such faults. Such a capability would be useful in applications in which timeliness is an important part of a dependability specification.

8 Conclusion

The AQuA project set out to demonstrate how the QuO architecture could provide the property of dependability of a service. This goal was attained by implementation of the AQuA system, released in conjunction with the QuO system. The AQuA release provides dependability of objects (clients as well as servers) by means of replication of the objects. The provision of dependability is highly controllable and adaptable at run time. This technology has been transferred outside the contractor and subcontractors, and has successfully been demonstrated both by the contractors and by an outside adopter. The AQuA effort had synergy with other related projects at BBN, providing additional payback during the contract period. Current ongoing related funded work provides an opportunity for further payback from the future use of the software and research base developed under AQuA.

References

[1] Adaptive quality of service for availability (AQuA). `<http://www.dist-systems.bbn.com/projects/AQuA/>`.

[2] M. Cukier, J. Ren, P. Rubel, D. E. Bakken, and D. A. Karr. Building dependable distributed objects with the AQuA architecture. In *Digest of Fast Abstracts Presented at the 29th Annual International Symposium on Fault-Tolerant Computing (FTCS-29)*, pages 17–18, June 1999.

[3] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. Berman, D. A. Karr, and R. E. Schantz. AQuA: an adaptive architecture that provides dependable distributed objects. In *Proc. of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 245–253, October 1998.

[4] Distributed systems department, bbn technologies. `<http://www.dist-systems.bbn.com/>`.

[5] The ensemble distributed communication system. `<http://www.cs.cornell.edu/Info/Projects/Ensemble/index.html>`.

[6] Object Management Group. CORBA success stories. <http://www.corba.org/>.

[7] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proc. of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98)*, April 1998.

[8] S. Maffeis. *Run-Time Support for Object-Oriented Distributed Programming*. PhD thesis, University of Zurich, 1995.

[9] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998.

[10] Performability engineering research group. <http://www.crhc.uiuc.edu/PERFORM/>.

[11] J. Ren, M. Cukier, P. Rubel, W. H. Sanders, D. E. Bakken, and D. A. Karr. Building dependable distributed applications using aqua. In *Proc. of the 4th IEEE Symposium on High Assurance Systems Engineering (HASE'99)*, November 1999. To appear.

[12] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W. H. Sanders, D. E. Bakken, and D. A. Karr. Proteus: A flexible infrastructure to implement fault tolerance in AQuA. In *Proc. of Seventh IFIP International Working Conference on Dependable Computing for Critical Applications*, January 1999.

[13] R. Schantz, J. A. Zinky, D. A. Karr, D. E. Bakken, J. Megquier, and J. P. Loyall. An object-level gateway supporting integrated-property quality of service. In *Proc. of the Second International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '99)*, May 1999.

[14] J. A. Zinky, D. E. Bakken, and R. E. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):55–73, April 1997.

DISTRIBUTION LIST

addresses	number of copies
AFRL/IFGA ATTN: TOM LAWRENCE 525 BROOKS ROAD ROME, NEW YORK 13441-4505	10
BBN TECHNOLOGIES 10 MOULTON STREET CAMBRIDGE, MA 92138	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/MLME 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739	1

AFRL/HESC-TDC 1
2698 G STREET, BLDG 190
WRIGHT-PATTERSON AFB OH 45433-7604

ATTN: SMDC IM PL 1
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

TECHNICAL LIBRARY D0274(PL-TS) 1
SPAWARSYSCEN
53560 HULL ST.
SAN DIEGO CA 92152-5001

COMMANDER, CODE 4TL000D 1
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

CDR, US ARMY AVIATION & MISSILE CMD 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

REPORT LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

ATTN: D'BORAH HART 1
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

ATTN: KAROLA M. YOURISON 1
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.